This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Using Client Puzzles to Mitigate Distributed Denial of Service Attacks in the Tor Anonymous Routing Environment

Nicholas A. Fraser, Douglas J. Kelly, Richard A. Raines, Rusty O. Baldwin, and Barry E. Mullins
Department of Electrical and Computer Engineering
Air Force Institute of Technology
{nicholas.fraser, douglas.kelly, richard.raines, rusty.baldwin, barry.mullins}@afit.edu

Abstract— A novel client puzzle protocol, the Memoryless Puzzle Protocol (MPP), is proposed and investigated. The goal is to show that MPP is a viable solution for mitigating distributed denial-of-service (DDoS) attacks in an anonymous routing environment. One such environment, Tor, provides anonymity for interactive Internet services. However, Tor relies on the Transport Layer Security (TLS) protocol, making it vulnerable to distributed denial-of-service (DDoS) attacks. Although client puzzles are often proposed as a solution to denial-of-service attacks, this research is the first to explore TLS DDoS attack mitigation in the Tor anonymous routing environment.

Using the MPP, the central processing unit (CPU) utilization and user-data latency measures are analyzed under four increasing DDoS attack intensities and four different puzzle probability distribution levels. For results, typical CPU utilization rates of 80-100% drop to below 70% signifying successful mitigation. Furthermore, even if a client only has a 30% chance of receiving a puzzle or the maximum puzzle strength is used, MPP effectively mitigates attacks. Finally, user-data latency decreases approximately 50% under large-scale attacks. Hence, the MPP is a suitable solution for increasing the robustness and reliability of Tor.

I. Introduction

A very common form of distributed denial of service (DDoS) attack in today's networks consist of a multitude of malicious clients flooding the network in the direction of a particular server to prevent legitimate clients from receiving such server service. Anonymous communications systems (ACS) that rely on an overlay network of servers to provide anonymity service are vulnerable to such DDoS attacks.

Tor [1] is one such ACS. Tor provides anonymous interactive Internet services like web, internet relay chat (IRC), and secure shell (SSH). Unfortunately, malicious zombies or bots [2] are more quickly controlling and managing IRC networks, or botnets, to launch DDoS attacks [3]. In 2003, for example, one honeypot research project saw 15,164 unique zombies form a large botnet within days [4]. In 2004, the Witty worm created 12,000 zombies within 45 minutes [5]. Botnets may send attack commands via IRC or peer-to-peer (P2P) channels [6]. The DDoS attacks use Transport Communications Protocol (TCP) SYN packet and User Datagram Protocol (UDP) flooding or more sophisticated techniques to degrade or eliminate legitimate user service. The common defenses of anomaly detection, rate-limiting and filtering may be ineffective under a full botnet attack [6]. A typical defense depends on a high level of end user internet security and privacy awareness. In addition, recent research into the Secure Overlay Services (SOS) [7] architecture, similar to Tor's onion routing architecture, has demonstrated the ability to proactively defend against a web denial of service attack.

In this paper, the goal is to show that MPP is a viable solution for mitigating DDoS attacks on Tor's interactive Internet services such as an anonymous web server. Client puzzles are investigated to protect against the inherent TLS protocol vulnerability and mitigate DDoS attacks. For Tor, puzzles are used to keep onion routers (OR) from completing the large number of decryption operations forced upon them during an attack. Of concern is the impact the client puzzle defensive technique has on OR CPU utilization and user-data latency. Since the OR pushes CPU load back onto clients during a DDoS attack, OR CPU utilization and latency are important parameters to assess puzzle effectiveness in maintaining service availability and preserving anonymity for legitimate clients.

In Section II, the implementation of MPP is delineated. In Section III, the analysis of OR CPU utilization and user-data latency using MPP under increasing attack intensities and differing probability distribution levels are examined. In Section IV, a succinct summary and conclusion are given.

Disclaimer: The views expressed are those of the author(s) and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **2007**	2. REPORT TYPE	3. DATES COVERED **00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE **Using Client Puzzles to Mitigate Distributed Denial of Service Attacks in the Tor AnonymousRouting Environment**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Electrical and Computer Engineering,Air Force Institute of Technology,Dayton,OH**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002055. Proceedings of the 2007 IEEE International Conference of Communications (ICC 2007) Held in Glasgow, Scotland on June 24-28, 2007. U.S. Government or Federal Rights License

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**8**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

II. PUZZLE IMPLEMENTATION

Client puzzles have been used as a cryptographic countermeasure to mitigate denial of service attacks [8]. In addition, client puzzles have been used to defend web servers running the Secure Sockets Layer (SSL) protocol [9]. Many others also used client puzzles [10, 11, 12, 13, 14, and 15]. In such schemes, the client solves a given puzzle before establishing a SSL/TLS session with a server. The client is forced to expend computational resources prior to requesting the server to carry out expensive operations to mitigate DDoS attacks. A description of the MMP implementation follows.

A. Description

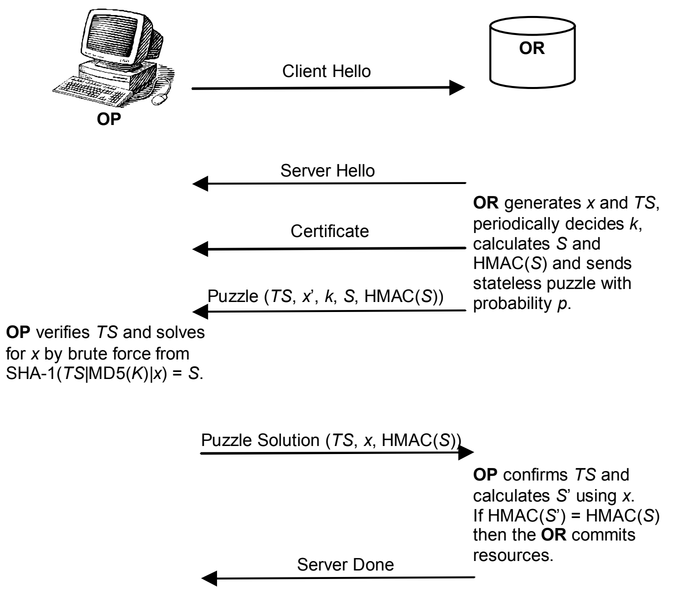
The proposed MPP for Tor's OR is shown in Figure 1. The MPP calls for an OR to construct a 512 bit string by concatenating (|) a timestamp (*TS*), a MD5 hash of its public key (*K*), and a random nonce (*x*). The string is hashed using SHA-1 to form a 160 bit string *S*. To complete the puzzle, the *k* lower ordered bits of *x* are set to zero to form *x'* and a keyed-hash message authentication code (HMAC) [16] of *S* is computed. HMAC is a secure hash function, like MD5 or SHA-1, wherein the user supplies a message, in this case *S*, and a secret key. If an OR is under attack and thus distributing puzzles, it probabilistically sends *TS*, *x'*, *k*, *S*, and HMAC(*S*) when a TLS connection request is received from either a malicious zombie or legitimate onion proxy (OP). Alleviating the workload of the OR and, to a lesser extend, legitimate clients is achieved by lowering the probability *p* of generating, sending and receiving a puzzle.

When an OP receives a puzzle, it verifies the *TS* is current before concatenating *TS*, a hash of the OR's public key, and *x'*. The OP solves the puzzle via a brute force approach by generating permutations for the *k* bits, hashing individual solutions, and comparing their hash values to *S*. Once the OP discovers *x*, it returns to the OR *TS*, *x*, and HMAC(*S*).

To verify the OP has submitted a correct solution, the OR confirms *TS* is current and then hashes (*TS*|*K*|*x*) = *S'* where *x* is supplied by an OP. The OR confirms it constructed the puzzle by verifying HMAC(*S'*) = HMAC(*S*). If a match is confirmed, the OR completes the TLS handshake.

MPP requires an OR execute two hash functions each time it supplies a puzzle and two hash functions when it verifies a solution. No state information is maintained even after a client has correctly solved a puzzle. A client can re-use a puzzle but only during its time window. Puzzle solutions cannot be pre-computed as the timestamp and a different *x* are part of the puzzle. Finally, by not sending the hash of an OR's public key, each OP retrieves this information locally and cannot be forced to solve another OR's puzzle.

A DDoS attack targeting Tor ORs using the TLS protocol is unavoidable, but it is possible to limit its effect. Additionally,



OP
OR
Client Hello
Server Hello
Certificate
Puzzle (TS, x', k, S, HMAC(S))
OR generates x and TS, periodically decides k, calculates S and HMAC(S) and sends stateless puzzle with probability p.
OP verifies TS and solves for x by brute force from SHA-1(TS|MD5(K)|x) = S.
Puzzle Solution (TS, x, HMAC(S))
OP confirms TS and calculates S' using x. If HMAC(S') = HMAC(S) then the OR commits resources.
Server Done

Figure 1: Proposed MPP
To mitigate a DDoS attack, the OR creates a specific strength puzzle based on threat/attack level and probabilistically sends the client a stateless Puzzle, the client replies with a brute force solution, and OR confirms before completing the TLS handshake and committing resources.

it is assumed an attacker likely possesses a collection of unwitting computer systems, like a botnet, to conduct their nefarious activity. These systems will have a wide range of computational capabilities and are not likely to be high-end processors specifically used to target Tor ORs. For this reason memory-based proof-of-work mitigation techniques [17] are not used. Also, Tor establishes multiple TLS connections at startup and refreshes these over time in the background. As a result, a user is assumed not to experience the TLS handshake delay given the 100 or fewer zombies in this study. However, if a reasonable-sized botnet of 10,000 zombies attacks, this assumption is unlikely to hold. Furthermore, legitimate clients are penalized if the malicious zombies are collectively more powerful. The implementation details are discussed next.

B. Implementation Details

Tor uses the OpenSSL library [18] for the TLS handshake. The 0.9.8 version of this C library is modified to incorporate client puzzles. A puzzle library is created and tested which contains the data structures for a puzzle, a puzzle solution and three functions: *create*, *solve*, and *verify*. These functions implement the MPP with only a few exceptions. First, timestamp verification is not implemented. Second, the client does not retrieve the public key of the server before solving a puzzle. Instead a hard-coded fake key is hashed by the client. The resulting hash is available only to *create* and *verify* functions. Finally, the *solve* function solves a puzzle by iterating through all the possible solutions. For example, if *k*=4, the order solutions attempted are 0x0, 0x1, 0x2, ..., 0xf. Other methods are possible like starting at 0xf and

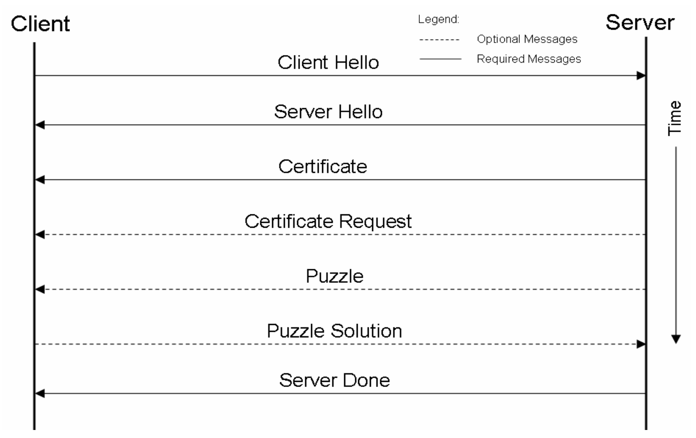


Figure 2: Modified TLS Handshake Sequence.
Server sets new SSL *puz* field to non-zero value and computes Tor-based puzzle strength and probabilistically distributes "Puzzle" message to clients. Client replies with "Puzzle Solution" message.

decrementing or randomly selecting a solution. In the latter case, failed solutions must be saved or solutions must be allowed to repeat. This is important because malicious individuals with access to the code could modify the *create* function so that all clients are forced to iterate through all possible solutions.

OpenSSL's TLS handshake implementation is modified with two new messages, the Puzzle and Puzzle Solution messages. The message sequence is shown in Figure 2.

Puzzle messages are only sent when a puzzle needs to be solved, i.e., when $k>0$. This is accomplished by adding a variable inside the SSL data structure. This integer variable, called *puz*, is set to zero when the SSL structure is initialized. A server application using the modified OpenSSL library sends a puzzle by setting *puz* to a non-zero value. How this value is calculated is left to the application.

Tor version 0.1.1.7-alpha is modified so ORs can assign puzzle strength and distribute puzzles during a TLS connection. Puzzle strength is determined by taking into account the overall OR CPU utilization. This is done in consideration of other services running on the OR.

These other services, along with the Tor service, might increase CPU utilization to the point where even a small attack could result in significant delays. The algorithm in *iostat* [19] is used to determine CPU utilization. To avoid incorrectly reacting to single spikes in CPU utilization, an average CPU utilization is used. More specifically, Tor's *run* function is executed every second. In this function, CPU utilization is polled and recorded in a circular array with five elements. When a connection request arrives, the average OR CPU utilization of the array is used to calculate the puzzle strength. The puzzle strength is the product of the average utilization percent and the maximum puzzle strength. This value can be easily changed by anyone with access to the code. The benefit of using an average is that an adversary is likely to find it difficult to hold a server's puzzle strength constant.

To further control puzzle distribution, a probabilistic decision variable is used. The variable is assigned a value each time a connection request occurs and is compared to a pre-configured threshold which determines if a puzzle should be sent. This feature is used to explore whether attacks can be mitigated without distributing a puzzle for every connection. Finally, Tor can easily be modified so that both client puzzles and the probabilistic decision feature can be turned off and on. The network topology setup is specified next.

C. Setup

The network topology as shown in Figure 3 consists of one OR, one client/server, and eight attack machines. The OR and client/server are 800 MHz machines with 256 MB RAM. The eight attackers are composed of five 1.7 GHz machines with 256 MB RAM while the remaining machines have 1.5 GHz, 2.4 GHz, and 2.0 GHz processors with 256 MB, 512MB, and 1GB RAM respectively. The number of attackers and their capability differences are limited by machine availability.

The attack program is developed using Tor's own functions. The user supplies the IP address of the victim, the number of children to be spawned, and a random number seed. Upon execution, the program spawns the correct number of children and uniquely seeds a random number generator for each process. Next, each process sleeps a random amount of time between zero and five seconds before establishing a TLS connection. Once established, the connection is terminated and the process again sleeps before again initiating a TLS connection.

By varying the number of processes in the attack program, the level (intensity) of the attack is varied. Moreover, the attack network accurately emulates a botnet: a master attack machine, when instructed to launch (conclude) an attack against the OR, signals the remaining seven machines to start (stop) attacking as well.

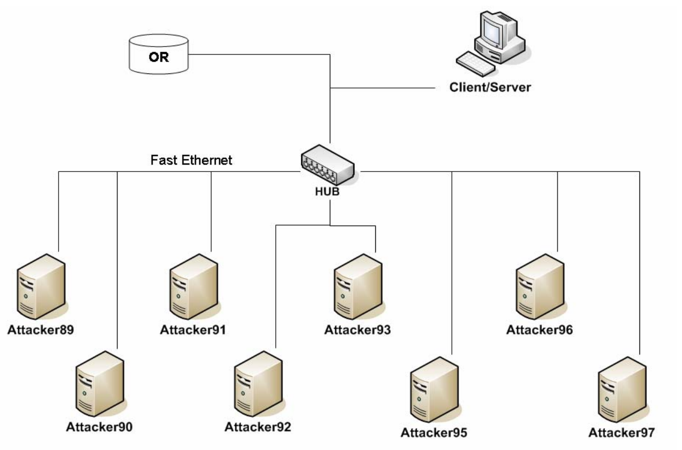


Figure 3: Network Topology.
One OR victim, five identical and three dissimilar attack clients emulating a botnet, and one client server connected via a 100Mbps Ethernet Hub.

All machines used during experiments are installed with Fedora Core 2. To measure average CPU utilization, the OR victim runs *vmstat* and sends the output to a file. To measure latency, a TCP client and server sends latency probes through the OR. When the client sends a packet, it inserts a timestamp. When the packet is received by the server, it records the client timestamp and the time the packet arrived at the server to a file. These timestamps are used to calculate the latency of the packet in microseconds. The analysis is explained next.

III. ANALYSIS

This section presents the performance analysis of the MPP. The analysis determines the effectiveness of the protocol using a puzzle distribution density function designed to defeat hash function DDoS attacks and limit the number of puzzle messages traversing the Tor network. Section A reveals the average CPU utilization performance results. Section B shows the user-data latency performance results.

A. CPU Utilization

To determine average CPU utilization, *vmstat* on the victim sends utilization statistics to a file every five seconds. Tor is configured to send a puzzle with probability *p*. Five experiments for each attack level (18, 38, 58, and 100 processes) are executed for seven minutes with one minute of idle time between each trial. By attacking for seven minutes and collecting utilization measurements every five seconds, sufficient data can be sampled while at the same time minimizing the amount of I/O required to record the data.

The victim has six configurations. The first configuration is an unmodified installation of Tor and OpenSSL. The remaining configurations use the modified Tor and OpenSSL but differ in the probability they will send a puzzle (0.0, 0.3, 0.5, 0.7, and 1.0).

First, the CPU utilizations from each of the four attack levels are measured to determine if an attack is actually taking place. The victim is a single OR; thus, the results are a best-case measurement. An attack is successful if CPU utilization exceeds 70%. This number is chosen because administrators often have “reserve” utilization available for unexpected events.

The average CPU utilization for an OR under attack from eight attack machines using the various process levels is shown in Figure 4. Puzzles are not distributed to mitigate the DDoS attack.

The solid line is the modified OpenSSL library and Tor application installation. The dashed line is the unmodified OpenSSL library and Tor application default installation.

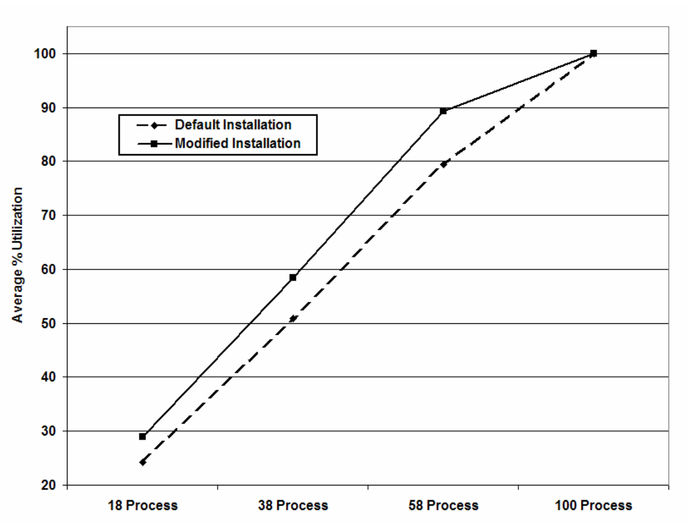


Figure 4: Attack Effects on Average CPU Utilization.
58 and 100 bots attacks successful (>70%).
Modified installation incurs overhead (≤10%).

As the Figure 4 shows and *t*-tests confirm, the CPU utilization is statistically greater at each level of attack intensity, with the exception of the 100 process attack.

For both installations, only the 58 and 100 process/bots attacks are considered successful since an average of at least 80% to 100% CPU utilization is achieved, respectively. The difference between the solid and dashed lines identifies the overhead induced when using modified Tor. The puzzle overhead increase never turns any of the non-attacks (18 and 38) into attacks, i.e., does not exceed 70%. Thus, the average CPU utilization due to overhead is approximately 7% and never exceeds 10%.

The average CPU utilization for each attack level when puzzles are distributed according to the various probabilities is shown below in Figure 5.

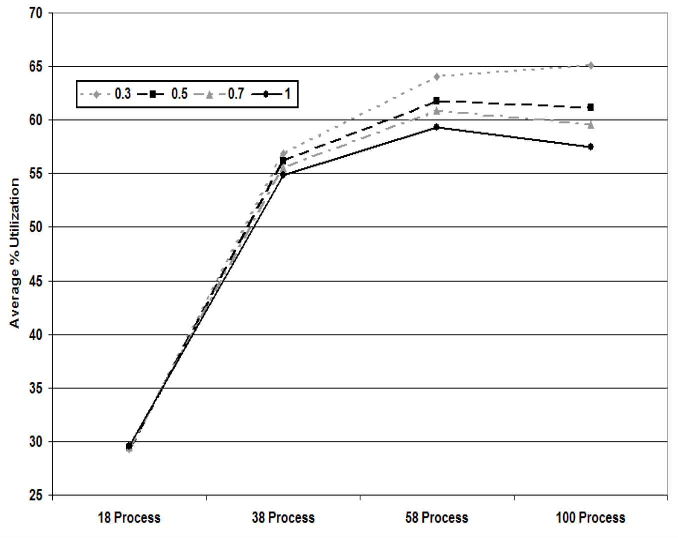


Figure 5: Average CPU Utilization.
58 and 100 bots DDoS attacks mitigated (<70%) for all probability levels.

Of foremost importance is that the 58 and 100 process/bots attacks are mitigated. In fact, they are mitigated so well, they can no longer be classified as attacks as all are below 70%. Furthermore, mitigation appears to improve as the puzzle distribution probability increases because the number of clients solving puzzles at any given time also increases. If clients are solving puzzles that means the server is not decrypting pre-master secrets. The only exception is when an 18 process attack is used. Such a result is likely due to weak puzzles created because of the weak attack.

Interestingly, a probability level of 0.3 is almost as effective at mitigating attacks as the other probability levels. This trend is likely due to the use of 30 as maximum puzzle strength. During the attack, a client will either receive a puzzle or not. If the client does not receive a puzzle, the request increases the CPU utilization. If the client does receive a puzzle, it is likely a strong one, i.e., greater than 25, because most clients have not received puzzles. Eventually, all attacking clients receive puzzles. Because the strengths are so strong the attack is effectively mitigated. For all attack and probability levels, an average utilization below 70% is achieved.

The final point of interest is the knee-shaped curve in relative mitigation performance between 58 and 100 process attacks. The 100 process attack appears to perform better than the 58 process attack. The reason is that too many attack processes execute on a single machine; thus, puzzles are solved at a slower rate. This causes an effect similar to solving a stronger puzzle. This experimental constraint was due to limited availability of attack computers.

CPU utilization given different puzzle strengths is shown in Figure 6. The most significant differences occur at the 100 process attack level. No puzzles achieve 100%, puzzle strength of 20 achieves 87% and puzzle strength of 30 achieves less than 60% CPU utilization, respectively.

A similar but more tightly bounded trend holds for the 58 attack process level. Hence, only increasing puzzle strength to

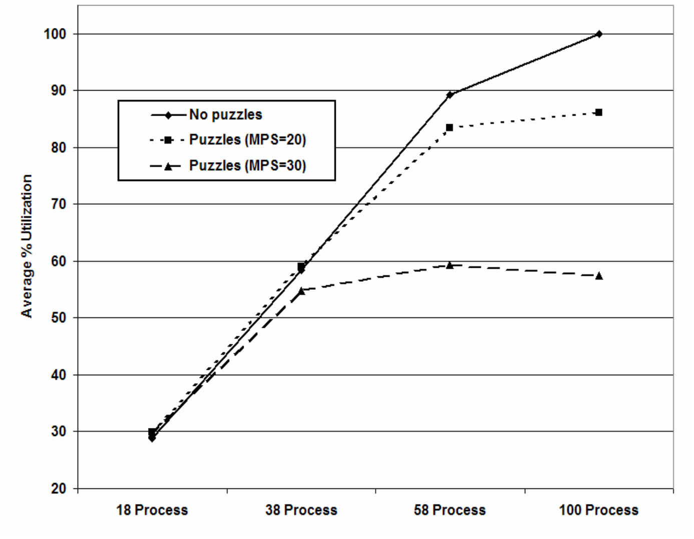


Figure 6: Puzzle Strength.
Maximum strength puzzle (MPS=30) mitigates attack (utilization <60%).

the maximum of 30 MPS mitigates a DDoS attack at the 58 and 100 process levels. The user data latency results are discussed next.

B. User Data Latency

To measure user-data latency, the victim is configured five ways. The first does not use client puzzles to protect against DDoS attacks. The other four use client puzzles at different probability levels of 0.3, 0.5, 0.7 and 1.0. Experiments are run for 13 minutes. Probes are sent at a rate of 10 per second. For the first three minutes the OR is not under attack. This allows a baseline to be established. After three minutes, an attack is initiated where 18, 38, 58, or 100 processes are distributed across the eight attack machines. Each attack lasts for seven minutes. The experiment ends after three additional minutes of sending probes.

User-data latency with increasing attack intensity is shown in Figure 7.

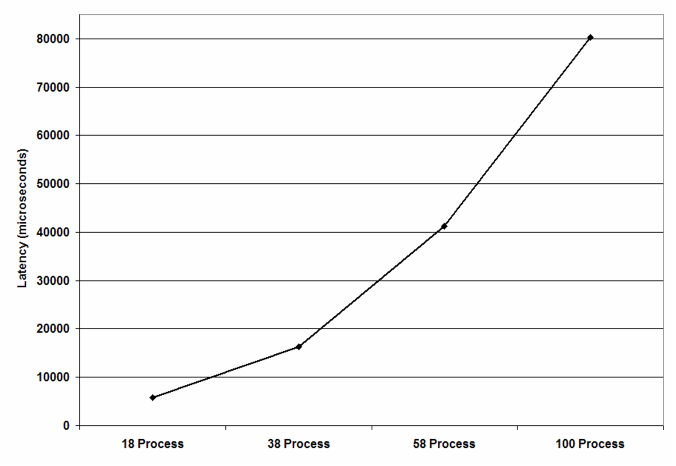


Figure 7: Attack Effects on Average Latency.
Larger attacks incur higher latency up to 80,000 microseconds.

As expected, the higher the attack intensity the slower the average user-data latency. The average latency of the highest level attack, 100 processes, is 80,000 microseconds. Achieving low-latency user-data delivery is a fundamental objective of Tor. When not under attack, user-data is delivered from the client to the server in approximately 1400 microseconds, i.e., almost instantaneously. Of course, data does not traverse multiple ORs. Plus, the propagation delay and network congestion are negligible. If client puzzles are used to mitigate an attack, the increase in latency is reduced. The average latency for each attack level at the different probability levels is shown in Figure 8. The 18, 38 and 58 process/bots attack latency measurements are not statistically different across probability levels. The 100 process/bots attack latency is statistically different across probability levels but the latency is minuscule and thus inconsequential.

However, the 100 process/bots attack latency is at least

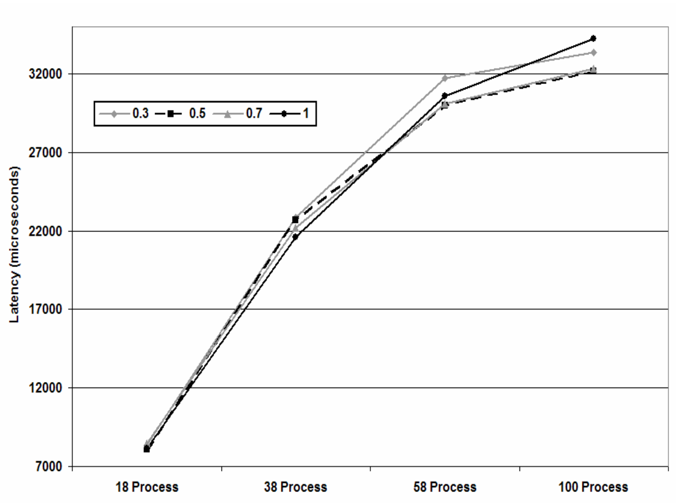


Figure 8: Average Latency.
Reduced latency for larger scale attacks for all probability levels. (>50% for 100 process and <50% for 58 process).

54% lower for probability 1.0 than in the Fig. 7 (37,000 versus 80,000 microseconds). The 58 process/bots attack latency is also lower (~30,000 versus 40,000 microseconds). An analysis of variance (ANOVA) reveals that neither the probability nor the interaction between attack level and probability contribute significantly to latency variation at the 95-percentile level. This neither strengthens nor weakens the assertion that distributing puzzles with a probability of 0.3 effectively mitigates the hash function attack and decreases the number of puzzle messages traversing the Tor network. So the differences in CPU utilization caused by probability changes are not enough to affect latency.

Hence, distributing puzzles 30% of the time is a feasible solution to both the decryption and hash function DDoS attacks. Although such a decision is not likely to significantly affect latency, it does make the integration of the MPP more appealing for Tor administrators. The conclusion is given next.

IV. Summary and Conclusion

In this paper, the novel client puzzle protocol MPP is proposed to mitigate TLS DDoS attacks in the Tor routing environment. Without MPP and under 58 and 100 botnet DDoS attacks, legitimate client anonymous service is adversely affected as indicated by an 80 to 100% CPU utilization rate on the onion router. However, with MPP and under similar DDoS attacks, legitimate client anonymous service improves as indicated by a less than 70% CPU utilization rate on the onion router for all four probability distribution levels. Additionally, with MPP and no attacks, only a 10% or less CPU utilization overhead is incurred on the onion router. Also, user-data latency decreases approximately 50% under the large-scale attacks. Hence, using MPP successfully mitigates malicious TLS DDoS attacks.

Moreover, even if a client only has a 30% chance of receiving a puzzle or the maximum puzzle strength is used, MPP effectively mitigates attacks. Also, the MPP puzzle probability distribution and the maximum puzzle strength are customizable to respond to a particular threat environment. Hence, MPP is a viable and flexible solution for mitigating DDoS attacks in an anonymous routing environment.

Extensions to this research might explore paths that mirror more closely the length and traffic volume across multiple Tor onion routers. This would provide additional insight into the suitability of MPP as a solution for increasing the robustness and reliability of Tor in preserving anonymity for interactive Internet services.

References

[1] Dingledine, Roger, N. Mathewson, P. Syverson. "Tor: The Second Generation Onion Router". *4th Proceedings of the 13th USENIX Security Symposium,* pp303-320, August 2004.
[2] R. Puri, "Bots & Botnets: An Overview," *Security & Privacy Magazine, IEEE* , vol.1, no.4, pp. 87-90, July-August 2003.
[3] T. Holz, "A short visit to the bot zoo [malicious bots software]," *Security & Privacy Magazine, IEEE*, vol.3, no.3, pp. 76-79, May-June 2005.
[4] B. McCarty, "Botnets: big and bigger," *Security & Privacy Magazine, IEEE* , vol.1, no.4, pp. 87-90, July-August 2003.
[5] C. Shannon and D. Moore, "The Spread of the Witty Worm," *IEEE Security & Privacy*, vol. 2, no. 4, pp. 46–50, 2004.
[6] G.P. Schaffer, "Worms and viruses and botnets, oh my! Rational responses to emerging Internet threats," *Security & Privacy Magazine, IEEE* , vol.4, no.3, pp. 52-58, May-June 2006.
[7] A.D. Keromytis, V. Misra, and D. Rubenstein, "Secure Overlay Services (SOS)", *Final Technical Report*, AFRL-IF-RS-TR-2004-236, Columbia University, August 2004.
[8] A. Juels, A. and J. Brainard, "Client Puzzles: A Cryptographic Defense against Connection Depletion," *5th Network and Systems Security Symposium*, pp. 151–165, 1999.
[9] D. Dean, A. Stubblefield, "Using Client Puzzles to Protect TLS," *Proceedings of the USENIX Security Symposium*, 2001.
[10] T. Aura, P. Nikander, and J. Leiwo, "DOS-Resistant Authentication with Client Puzzles," Lecture Notes in Computer Science, vol. 2133, 2001.
[11] J. Leiwo, T. Aura, and P. Nikander, "Towards Network Denial of Service Resistant Protocols," in *SEC*, pp. 301–310, 2000.
[12] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach, "Security for Peer-to-Peer Routing Overlays," in *Proceedings of OSDI*, December 2002.
[13] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately Hard, Memory-bound Functions," 2003.
[14] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Lecture Notes in Computer Science*, vol. 2009, pp. 46+, 2001.
[15] C. Castelluccia, E. Mykletun, G. Tsudik, "Improving Secure Server Performance by Re-balancing SSL/TSL Handshakes," *ASIACCS'06*, 2006.
[16] H. Krawczyk, R. Canetti, M. Ballare. "Request For Comments (RFC) 2104 – HMAC: Keyed-Hashing for Message Authentication". URL http://www.faqs.org/rfcs/rfc2104.html, February 1997
[17] Abadi, Martin, M. Burrows, M. Manasse, T. Wobber. "Moderately hard, memory-bound functions". *ACM Trans. Inter. Tech*., 5(2):299-327, 2005.
[18] D.G. Andersen. "Mayday: Distributed Filtering for Internet Services". *4th Usenix Symposium on Internet Technologies and Systems*., Seattle WA, USA, March 2003.
[19] S. Godard. *iostat*. http://perso.wanadoo.fr/sebastien.godard/, January 2005.